I. J. Computer Network and Information Security, **2013, 9, 51-57**
Published Online July 2013 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2013.09.07

Improved Trial Division Technique for Primality Checking in RSA Algorithm

Kumarjit Banerjee, Satyendra Nath Mandal, Sanjoy Kumar Das
Systems Engineer- Tata Consultancy Services, Dept. of I.T, Kalyani Govt. Engg College, Kalyani, Nadia (W.B), India
University of Kalyani, Nadia (W.B), India
kumarjit.banerjee@tcs.com, satyen_kgec@rediffmail.com, dassanjoy0810@hotmail.com

Abstract — The RSA cryptosystem, invented by Ron Rivest, Adi Shamir and Len Adleman was first publicized in the August 1977 issue of Scientific American. The security level of this algorithm very much depends on two large prime numbers. To check the primality of large number in personal computer is huge time consuming using the best known trial division algorithm. The time complexity for primality testing has been reduced using the representation of divisors in the form of 6n±1. According to the fundamental theorem of Arithmetic, every number has unique factorization. So to check primality, it is sufficient to check if the number is divisible by any prime below the square root of the number. The set of divisors obtained by 6n±1 form representation contains many composites. These composite numbers have been reduced by 30k approach. In this paper, the number of composites has been further reduced using 210k approach. A performance analysis in time complexity has been given between 210k approach and other prior applied methods. It has been observed that the time complexity for primality testing has been reduced using 210k approach.

Index Terms — Improved Trial Division, RSA Algorithm, Primality Checking, Pseudo primes, 210k Approach

I. INTRODUCTION

The requirements of information security within an organization have undergone two major changes in the last few decades. With the introduction of the computer the lead of automated tools for protecting files and other information stored on the computer became evident, especially the case for a shared system. The RSA algorithm is the most popular and proven asymmetric key cryptographic algorithm [1]. The importance of asymmetric key cryptography is that, the private key does not to be shared on the network. Only the public key is shared [10]. A more formal definition of asymmetric cryptosystem may be given as: A cryptosystem consisting of a set of enciphering transformations {Ee} and a set of deciphering transformations {Dd} is called a Public-key Cryptosystem or an Asymmetric Cryptosystem if, for each key pair (e, d), the enciphering key e, called the public key, is made publicly available, while the deciphering key d, called the private key, is kept secret. The cryptosystem must satisfy the property that it is computationally infeasible to compute d from e [16]. The RSA algorithm first requires two sufficiently large primes to be chosen [11]. For this purpose the primality tests on the numbers has to be computed. The trial division algorithm has been considered which can be used both for primality testing and factorization of numbers [12]. The time complexity for the algorithm has been reduced each time with each new approach [4] from slightly higher than ½√n to 1/3√n to 4/15√n with the application of 30k method [9]. On a modern workstation, and very roughly speaking, numbers that can be proved prime via trial division in one minute do not exceed 13 decimal digits. However the time is reduced and for an 18 decimal digits number, it takes about 1 hour 5mins. This is a considerable gain. Also the time is further reduced to 50mins (not considering the time for finding the multiplicative order) [7]. The time is drastically reduced to less than 3 mins.

The Fermat's method is very efficient in finding the factors of a number which are close to the square root of the number. Thus the worst case for trial division is the best case for Fermat's method [15]. In this regard, experiments prove that Fermat's method is inferior to trial division for primality testing as the square of the difference increase rapidly. The aim of using Miller-Rabin algorithm is to check the primality of a given number using trial division with the set of primes instead of the set of pseudo primes [3] where the number of composites increases exponentially. This however also proved to be ineffective as an extra overhead occurs in separating the primes from composites. On the other hand, Miller-Rabin algorithm may be used to first check the given number is prime or not. If the given number passes the Miller-Rabin test, then it should be checked with trial division. The reason is that, if a number fails the Miller-Rabin test then the number is surely a composite number. But however if it passes the test the number may be prime or composite. It is in this case that must be checked with trial division.

In this paper, the time complexity of the best known trial division algorithm so far, has been further reduced by 210k approach. In 210k approach, the divisors in the set of pseudo primes have been represented by a set of linear polynomials. The coefficients of the polynomials have been computed using the GCD of the numbers below square root of the given number to 210. The reason behind it is that the ratio of φ(n)/n is lowest for 210 compared to other numbers below 210. A table has been indicated that the number of possible primes below a given number [13]. This approach has reduced the number of pseudo primes that is the number of pseudo primes are close to the number of primes shown in Table 1[13]. Finally, a comparison has been made between 210k method and other applied methods. After finding the primes, RSA algorithm has been implemented and the algorithm has been applied on different types of files with different sizes.

This paper is divided into the following parts. Article 1 is the introduction. Article 2 describes the RSA algorithm. Article 3 provides the primality checking methods and how they can be implemented efficiently. Article 4 describes the facts and algorithms and also the algorithm for finding square root of a large number expressed as string. The Article No. 5 is the implementation. This part is the vital part of the project as it distinguishes the various algorithms used based on the time complexity. Article 6 is the part for future works based on the conclusions from this paper. A list of references is provided at the end.

II. RSA ALGORITHM

The RSA algorithm involves three steps: key generation, encryption and decryption. Each step is described below in sections *A*, *B* and *C*.

A. KeyGeneration

RSA involves a public key and a private key. The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted using the private key. The keys for the RSA algorithm are generated the following way:

Choose two distinct prime numbers p and q. For security purposes, the integers p and q should be chosen uniformly at random and should be of similar bit-length. Prime integers can be efficiently found using a Primality test. Compute $n = p*q$. n is used as the modulus for both the public and private keys. Compute the totient: $\varphi(n) = (p\text{-}1)*(q\text{-}1)$. Choose an integer e such that $1<e< \varphi(n)$, and e and $\varphi(n)$ are coprime. e is released as the public key exponent. Choosing e [2, 6] having a short addition chain results in more efficient encryption. Determine d (using modular arithmetic) which satisfies the congruence relation $d*e \equiv 1(\text{mod } \varphi(n))$. d is kept as the private key exponent. The public key consists of the modulus n and the public (or encryption) exponent e. The private key consists of the modulus n and the private (or decryption) exponent d which must be kept secret.

B. Encryption

Alice transmits her public key (n,e) to Bob and keeps the private key secret. Bob then wishes to send message M to Alice. He first turns M into an integer $0<m< n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext c corresponding to: $c \equiv m^e(\text{mod } n)$. This can be done quickly using the method of exponentiation by squaring. Bob then transmits c to Alice [17].

C. Decryption

Alice can recover m from c by using her private key exponent d by the following computation: $m \equiv c^d(\text{mod } n)$. Given m, she can recover the original message M by reversing the padding scheme.

The above decryption procedure works because:

$$m \equiv (m^e)^d(\text{mod } n) \equiv m^{ed}(\text{mod } n).$$

Now, since $e*d=1+k* \varphi(n)$,

$$m^{ed} \equiv m^{1+k*\varphi(n)} \equiv m*(m^k)^{\varphi(n)} \equiv m(\text{mod } n)$$

The last congruence directly follows from Euler's theorem when m is relatively prime to n. By using the Chinese remainder theorem it can be shown that the equations hold for all m. This shows that the original message is retrieved:

$$c^d \equiv m(\text{mod } n).$$

D. RSA Conjecture

The famous RSA conjecture states that Cryptanalyzing RSA must be as difficult as factoring.

However, there is no known proof of this conjecture, although the general consensus is that it is valid. The reason for the consensus is that the only known method for finding d given e is to apply the extended Euclidean algorithm to e and φ(n). Yet to compute φ(n), we need to know p and q, namely, to cryptanalyze the RSA cryptosystem, we must be able to factor n. To break RSA, or rather to recover the plaintext from decrypted text, factorization may not be the only possible way. There are several kinds of attacks on RSA. For example an instance of chosen ciphertext attack demons treated in paper [18]. It is common to take the ascii value of the text as plaintext and encrypt it. The attacker may not have to know the private key or do any factorization on n and. He simply runs a loop and finds the plain text. The time complexity is shown. Another common attack is the short private key exponent attack. In this case, the value of d is chosen is small, so that again the attacker can run a loop and decrypt the message. The attacks which are discussed, the former one may be prevented by using efficient padding and the second one may be solved using a short public key. The variable padding scheme not only removes the weakness of chosen cipertexts, but also protects the message from another kind of attack known as the frequency attack. RSA as it is known that for a given plaint text it will produce the same cipher text for a given pair of (n, e). The idea for variable n-padding is to completely remove the frequency attack. For the second kind of attack, along with choosing large primes p and q, one must also choose e to be small. This is because, for the fact that e and d satisfies the relation e*d=1+k* φ(n).

If one chooses e to be small then, d will be at the order of φ(n)/e. As n is large so is φ(n). Thus it makes d large.

Both the types of attack do not involve factorization of n. Now, coming to the difficulty of factorization of n, the primes of RSA algorithm must be chosen carefully so that it will be difficult to factor. The obvious thought would com to choose the two primes as close as possible. But it is also very much prone to factorization using Fermat's method. The number which is hard to factor using trial division is as simple for Fermat's method. Fermat's method acts backwards in comparison to trial division which acts forward. The next section describes how Fermat's method works.

III. PRIMALITY CHECKING METHODS

A. The Fermat's method

If one can write n in the form $a^2 - b^2$, where a, b are nonnegative integers, then one can immediately factor n as $(a + b)(a - b)$. If $a - b > 1$, then the factorization is nontrivial [14]. Further, every factorization of every odd number n arises in this way. Indeed, if n is odd and $n = uv$, where u, v are positive integers, then $n = a^2 - b^2$ with $a = ½ (u + v)$ and $b = ½|u - v|$.

For example, consider $n = 8051$. The first square above n is 8100 = 902, and the difference to n is 49 = 72. So 8051 = (90 + 7)(90 − 7) = 97 · 83. To formalize this as an algorithm, we take trial values of the number a from the sequence ceil(√n2), *ceil(*√n2)+ 1, . . . and check whether a^2-n is a square. If it is, say b^2, then we have $n = a^2-b^2 = (a+b)(a-b)$.

Each iterations of Fermat's method reduces the upper bound for trial division by $\sqrt{n} - \sqrt{(a^2-n)}$. This reduction in the upper bound means for less complexity in trial division as computing the square root every time is very very costly operations. The method is effective for factorization but poor for primality checking because it will always execute the worst case scenario of this algorithm.

B. Miller-Rabin Algorithm

The Miller–Rabin primality test or Rabin–Miller primality test is a primality test: an algorithm which determines whether a given number is prime [5]. Its original version, due to Gary L. Miller, is deterministic, but the determinism relies on the unproven generalized Riemann hypothesis; Michael O. Rabin modified it to obtain an unconditional probabilistic algorithm. The pseudo code for the algorithm is presented below.

Algorithm for the Miller-Rabin Probabilistic Primality Test

```
Miller-Rabin(n,t)
INPUT: An odd integer n > 1 and a positive
    security parameter t
OUTPUT: the answer "COMPOSITE" or
    "PRIME"

Write n - 1 = 2sr such that r is odd
    Repeat from 1 to t
    Choose a random integer a which satisfies
     1 < a < n - 1

    Compute y = ar mod n
    If y > 1 and y < n-1 then DO

    j := 1
    WHILE j < s and y < n - 1 then DO

    y := y2 mod n
    if y = 1 then return("COMPOSITE")
    j := j + 1

  if y < n - 1 then return("COMPOSITE")
  return("PRIME").
```

IV. FACTS DATA AND ALGORITHMS

A. Fact 1:

Every prime number is any of the either forms 30k+1, 30k+7, 30k+11, 30k+13, 30k+17, 30k+19, 30k+23, 30k+29 apart from 2, 3, 5. The above fact is true for 30k approach. This set of linear polynomials can be reduced by using the below set of polynomials in the case of 210k approach. Every prime number is any of the either of the below forms apart from 2, 3, 5 and 7.

210k+11, 210k+13, 210k+17, 210k+19, 210k+23, 210k+29, 210k+31, 210k+37, 210k+41, 210k+43, 210k+47, 210k+53, 210k+59, 210k+61, 210k+67, 210k+71, 210k+73, 210k+79, 210k+83, 210k+89, 210k+97, 210k+101, 210k+103, 210k+107, 210k+109, 210k+113, 210k+121, 210k+127, 210k+131, 210k+137, 210k+139, 210k+143, 210k+149, 210k+151, 210k+157, 210k+163, 210k+167, 210k+169, 210k+173, 210k+179, 210k+181, 210k+187, 210k+191, 210k+193, 210k+197, 210k+199, 210k+209

Proof: From the division algorithm, any integer can be expressed in any of the forms.

For a given number n

$$n = q*d+r$$

where q is the quotient, d is the divisor and r is the remainder. Here d is prime. So the set of numbers generated as a result of this equation is the set of pseudo primes for d if and only if gcd (d,r)=1.

The set of numbers which cannot be expressed as an explicit product of two numbers among the above numbers, are the set of pseudo primes. The set of primes except 2, 3, 5 and 7 is a subset of the pseudo primes.
Hence *proved*.

B. Choosing the value for which set of pseudo primes are generated.

The number 30 is so chosen so that the ratio of the number of elements of the set of pseudo primes to the number is least. Another example of such a number is 12 for which the number of elements of pseudo primes is 4. However 4/12 =1/3 =2/6 which is the same as the primes expressed as 6k±1. Choosing 12 thus gives us no extra advantage. But choosing 30, the ratio is 8/30=4/15<1/3. For the number 210, this ratio is reduced to 4/15*6/7= 8/35. This advantage in turn reduces the time complexity shown in the results (Section V). The table [Table 1] shows the number of primes below x defined by the function pi(x).

The data for the below table is taken from the internet and is assumed to be correct. Further calculations are made assuming the correctness of the data provided in table I [13].

TABLE I. TAKEN FROM HTTP://PRIMES.UTM.EDU/HOWMANY.SHTML ON 12.09.2012

Sl No	x	pi(x)
1	1×10^{1}	4
2	1×10^{2}	25
3	1×10^{3}	168
4	1×10^{4}	1,229
5	1×10^{5}	9,592
6	1×10^{6}	78,498
7	1×10^{7}	664,579
8	1×10^{8}	5,761,455
9	1×10^{9}	50,847,534
10	1×10^{10}	455,052,511
11	1×10^{11}	4,118,054,813
12	1×10^{12}	37,607,912,018
13	1×10^{13}	346,065,536,839
14	1×10^{14}	3,204,941,750,802
15	1×10^{15}	29,844,570,422,669
16	1×10^{16}	279,238,341,033,925
17	1×10^{17}	2,623,557,157,654,233
18	1×10^{18}	24,739,954,287,740,860
19	1×10^{19}	234,057,667,276,344,607
20	1×10^{20}	2,220,819,602,560,918,840
21	1×10^{21}	21,127,269,486,018,731,928
22	1×10^{22}	201,467,286,689,315,906,290
23	1×10^{23}	1,925,320,391,606,803,968,923
24	1×10^{24}	18,435,599,767,349,200,867,866

C. Algorithm for Square root

1. Take the input number as String
2. Compute the length of the number.
3. If the length is odd go to step 5.
4. If the length is even go to step 6.
5. Compute the square root of the first digit using the available square root method and add (lengh-1)/2 zeros and next go to step 7.
6. Compute the square root of the first two digits using the available square root method and add (length – 2)/2 zeros and next go to step 7.
7. From the second place from left 1 is added and multiplied by itself to check whether it is greater than the given number. Once it is greater the previous number is restored and manipulation is done for the next digit until the units' place arrives.

D. Improved Trial division

1. Take the given number as java.lang.math.BigInteger [8].
2. Compute the Miller-Rabin test as described the algorithm 3.2
3. If the number fails the Miller-Rabin test return composite.
4. If the number passes the Miller-Rabin test go to step 5.
5. Compute the modulo with successive values of the pseudo prime sets and increasing the count.
6. Each time the modulo is considered below the lower bound of square root of n.
7. If it is zero return composite else return prime.

V. RESULTS

A. Number of composites for each Approach

This section details the results obtained by the use of the above mentioned technique. The numbers of composites are computed based the fact obtained from Table 1. Finally the time required for each are computed and depicted in the table 3. Comparisons are also made showing the efficiency of the above mentioned technique.

B. RSA Example

The following example demonstrates the RSA algorithm.

P=45310159786437928331, q= 70228961500618843931,
n=p*q =
3182085467228637408294035624555852309161
$\varphi(n) = (p-1)*(q-1)$ =
3182085467228637408178496503268795536900
Choose e = 757, such that d =
2080756018861526442600205771622263924393

Encryption
Plain text
Improved Trial division with other methods for primality checking in RSA Algorithm.
Encrypted text

2187717888271082115657598202822593854486
2094957440981465180005425290113127677599
2441552423251817602653936715377315678684
971800371121278542986050073332786589738
2884037197356820817095270609196309664603
2207268391862251396880712252389204064261
772681732471832165569423765746037262090
2802959384322644829963951923881409550112
803572029218039783303683482293382425 33
1829647827363985483809677734551794793752
971800371121278542986050073332786589738
1318269330279187339217980874276166483343
1604643270511594169231922413874563065262
761856378431955540348517493691780746615
803572029218039783303683482293382425 33

TABLE II
NUMBER OF COMPOSITES IN THE SET OF PSEUDO PRIMES FOR 6K AND 30K VS. 210K APPROACH

Sl No	6k Method No of Composites	30k Method No of Composites	210k Method No of Composites
1	1	1	0
2	10	4	1
3	167	101	64
4	2106	1440	1060
5	23743	17077	13269
6	254837	188171	150077
7	2668756	2002090	1621139
8	27571880	20905214	17095691
9	282485801	215819135	177723898
10	2878280824	2211614158	1830661778
11	29215278522	22548619856	18739094905
12	295725421317	229058754651	190963516557
13	2987267796496	2320601129830	1939648748879
14	30128391582533	23461724915867	19652201106344
15	303488762910666	236822096244000	198726858148763
16	3054094992299410	2387428325632744	2006475944680364
17	30709776175679102	24043109509012436	20233585699488628
18	308593379045592475	241926712378925809	203831474283687715
19	3099275666056988728	2432608999390322062	2051656618437941111
20	31112513730772414495	24445847064105747829	20636323254581938306
21	312206063847314601407	245539397180647934741	207444159085409839504
22	3131866046644017427045	2465199379977350760379	2084246999024969807999
23	31408012941726529364412	24741346275059862697746	20931822465536053173938
24	3148977335659841324654 69	248231066899317465798803	210135828804079370560709

2802959384322644829963951923881409550112
1318269330279187339217980874276166483343
22072683918622513968807122523892040642611131826
933027918733921798087427616648334 3
2810315011685597722786164651368617218860
1318269330279187339217980874276166483343
2884037197356820817095270609196309664603
2675437138876851168416256073695826170349
803572029218039783303683482293382425 33
1654055379167895797578384756844735038268
1318269330279187339217980874276166483343
789293103987398374790245357507314232978
1779855900031666726072016326828737105399
803572029218039783303683482293382425 33
2884037197356820817095270609196309664603
789293103987398374790245357507314232978
1779855900031666726072016326828737105399
772681732471832165569423765746037262090
971800371121278542986050073332786589738
803572029218039783303683482293382425 33
2094957440981465180005425290113127677599
772681732471832165569423765746037262090
789293103987398374790245357507314232978
1779855900031666726072016326828737105399
2884037197356820817095270609196309664603
2802959384322644829963951923881409550112
2810315011685597722786164651368617218860
803572029218039783303683482293382425 33
897082554088968940791461829015501971424
2884037197356820817095270609196309664603
971800371121278542986050073332786589738
803572029218039783303683482293382425 33
2441552423251817602653936715377315678684
971800371121278542986050073332786589738
1318269330279187339217980874276166483343
2094957440981465180005425290113127677599
1604643270511594169231922413874563065262
761856378431955540348517493691780746615
1318269330279187339217980874276166483343
789293103987398374790245357507314232978
2129855640900820856509773396779314456537
803572029218039783303683482293382425 33
2430399895991345806690086588344484585695
1779855900031666726072016326828737105399
772681732471832165569423765746037262090
2430399895991345806690086588344484585695
2677467002691067105871255981026629326523
1318269330279187339217980874276166483343
2675437138876851168416256073695826170349
1443920989383541629663026420844857552213
803572029218039783303683482293382425 33
1318269330279187339217980874276166483343
2675437138876851168416256073695826170349
803572029218039783303683482293382425 33
2520748065686772398889309293290726794198
985918902304417468083655066925617894933
1708403870386847777879653029653886535352
803572029218039783303683482293382425 33
1708403870386847777879653029653886535352
761856378431955540348517493691780746615
1443920989383541629663026420844857552213
2884037197356820817095270609196309664603
971800371121278542986050073332786589738
1318269330279187339217980874276166483343
789293103987398374790245357507314232978
1779855900031666726072016326828737105399
2094957440981465180005425290113127677599
1802345906757163408144528490262651983897

Decrypted text

Improved Trial division with other methods for primality checking in RSA Algorithm.

C. Time taken in primality checking and time taken for encryption/decryption

TABLE III
COMPARISON BETWEEN TIMES FOR PRIME CHECK OF THE PREVIOUS ALGORITHM AND WITH THIS DETERMINISTIC APPROACH (TIME VERY LARGE MOSTLY GREATER THAN 2 HOURS IS NOT MENTIONED)

Digits	Prime	[1]	[2]	6k	30k	210k
3	101	< 1	< 1	<1	<1	<1
3	751	< 1	< 1	<1	<1	<1
4	1201	< 1	< 1	<1	<1	<1
4	9091	< 1	< 1	<1	<1	<1
5	10753	< 1	< 1	<1	<1	<1
5	76801	< 1	< 1	<1	<1	<1
6	160001	< 1	< 1	<1	<1	<1
6	980801	< 1	< 1	<1	<1	<1
7	1146881	< 1	< 1	<1	<1	<1
7	9011201	< 1	< 1	<1	<1	<1
8	12600001	< 1	< 1	<1	<1	<1
8	99328001	< 1	< 1	<1	<1	<1
9	104857601	< 1	< 1	<1	<1	<1
9	756100001	< 1	< 1	<1	<1	<1
10	1027200001	< 1	< 1	<1	<1	<1
10	9524994049	1	< 1	<1	<1	<1
11	10256250001	1	< 1	<1	<1	<1
11	97656250001	2	1	<1	<1	<1
12	100907200001	2	1	<1	<1	<1
12	947147262401	3	2	<1	<1	<1
13	1079916250001	5	2	<1	<1	<1
13	9982699110401	8	6	<1	<1	<1
14	12123750000001	10	9	<1	<1	<1
14	87770788000001	25	25	1	< 1	< 1
15	101702694862849	53	54	2	1	1
15	944377409044481	113	88	6	4	4
16	1136591040000001	127	106	6	4	4
16	9502720000000001	305	257	19	14	12
17	12136000000000001	702	518	22	16	14
17	95348273971200001	1410	1110	63	47	40
18	100663296000000001	1630	1126	65	48	41
18	908800000000000001	3990	2980	198	146	125
19	1000000000000000003	~	~	208	153	132
19	9999999999999999961	~	~	693	502	426
20	10000000000000000051	~	~	728	506	429
20	99999999999999999989	~	~	2861	2120	1816
21	100000000000000000039	~	~	2969	2139	1891
21	999999999999999999899	~	~	~	~	6901
22	1000000000000000000117	~	~	~	~	6902

TABLE IV
TIME TO ENCRYPT AND DECRYPT DIFFERENT TYPES OF FILES

Size of file	Type of file	Time to encrypt in secs	Time to decrypt in secs
1 KB	txt	2	1
10 KB	txt	12	6
14.5 KB	gif	18	8
44.1 KB	mp3	51	22
100 KB	doc	97	42
100 KB	txt	123	54
121 KB	pdf	150	65
427 KB	jpg	590	226
1 MB	doc	1054	450
1 MB	txt	1225	595
47.7 MB	VOB	6325	2787

VI CONCLUSIONS AND FUTURE WORKS

In this paper, the number of composites is reduced in the set of pseudo primes, and the numbers have been produced by this approach is almost close to the number of exact primes with a given range of numbers. As the number of composites has been reduced, the performance of the algorithm has been improved in terms of time complexity. But the primes cannot be eliminated completely. It should be investigated the growth pattern of primes within the pseudo prime set. It will help to guide us the approach for choosing the optimized polynomial set for generating pseudo primes. Steps should also be taken to reduce the complexity of the exiting program such as suppressing logs and intermediate steps to calculate and compute the time complexity of the algorithm. It also needs investigation for any improvement may be done regarding the design pattern of the existing program or software used to calculate the time.

REFERENCES

[1] Ron L. Rivest, Adi Shamir, and Len Adleman, "A method for obtaining digital Signatures and public-key cryptosystems", Communications of the ACM 21 (1978), pp 120-126.

[2] Boneh and Durfee, "Cryptanalysis of RSA with private key d less than n0.292", IEEETIT: IEEE Transactions on Information Theory, Volume 46, Issue 4, Jul 2000 pp. 1339 – 1349.

[3] C. Pomerance, J. L. Selfridge and Wagstaff, Jr., S. S., "The pseudoprimes to 25•109", Math. Comp., 35:151 (1980) pp. 1003–1026.

[4] Mandal N. Satyendra, Banerjee Kumarjit, Maiti Biswajit, Palchaudhury J. , "Modified Trail division for Implementation of RSA Algorithm with Large Integers", Int. Journal Advanced Networking and Applications Volume: 01, Issue: 04, Pages: 210-216 (2009)..

[5] Michael O. Rabin, "Probabilistic algorithm for testing primality", Journal of Number Theory, Volume 12 Issue no. 1, pp. 128–138 (1980)..

[6] M. Wiener, "Cryptanalysis of short rsa secret exponents", IEEE Transactions on Information Theory 36 (1990), pp.553-558.

[7] Banerjee Kumarjit, Mandal N. Satyendra, Palchaudhury J, Banerjee Abhishek, "A Deterministic Approach in Trial Division with Pseusdoprimes for RSA Implementation with Large Numbers", IJCSES International Journal of Computer Sciences and Engineering Systems, Vol.5, No.1, January 2011.

[8] Guicheng Shen, Bingwu Liu, Xuefeng Zheng, "Research on Fast Implementation of RSA with Java", Nanchang, P. R. China, May 22-24, 2009, pp. 186-189.

[9] Banerjee Kumarjit, Mandal Satyendra Nath, Das Sanjoy Kumar, "A Comparative Study of Different Techniques for Prime Testing in Implementation of RSA", ISSN No. 2229-6611, IEMCON 2012, Vol. 2, No. 2, pp 42-47.

[10] David M. Burton, "Elementary Number Theory (2nd ed)", pp 175-181, Universal Books Stall, New Delhi, (2004).

[11] Whitfield Diffie and Martin E. Hellman, "New directions in cryptography", IEEE Transactions on Information Theory IT-22, no. 6, 1976, pp644-654.

[12] Tatsuaki Okamoto and Shigenori Uchiyama, "A new public key cryptosystem as secure as factoring", Lecture notes in Computer Science 1403 (1998), 308-318. MR 1 729 059.

[13] The Prime Pages-prime number research, records and resources, available at http://primes.utm.edu/howmany.shtml last accessed 12.09.2012.

[14] Richard Crandall and Carl Pomerance, "Prime Numbers A Computational (Second Edition)", pp. 83-113 and pp 173-179 and pp 225-227,Springer ISBN-10: 0-387-25282-7,New York(2005).

[15] Song. Y Yan, "Cryptanalytic Attacks on RSA",pp 55-93 and pp 149-187,Springer ISBN-13: 978-0-387-48741-0,New York (2008).

[16] Hinek M. Jason, "Cryptanalysis of RSA and Its Variants", pp 8-10 and pp 17-23, CRC Press, New York (2010).

[17] Mollin A. Richard, "RSA and Public-Key Cryptography", pp 53-108, CRC Press LLC, New York (2003).

[18] Mandal N. Satyendra, Banerjee Kumarjit, Palchaudhury J., Implementation of RSA Algorithm with variable n-padding technique and Miller-Rabin test for Auto-generated Large keys from System Date, Proceedings of the 5th National Conference; INDIACom-2011.

Satyendra Nath Mandal (23/10/1975) has received his B.Tech & M.Tech in Computer Science & Engineering from university of Calcutta, West Bengal India. He is now working as an assistant professor in department of Information Technology at Kalyani Govt. Engg. College , Kalyani , Nadia, West Bengal, India. His field of research areas includes cryptography & network Security, fuzzy logic, Artificial Neural Network, Genetic Algorithm etc. He has about 35 research papers in national and International conferences. His six research papers have been published in International journal.

Kumarjit Banerjee (27/08/1986) has received his B. Tech degree in Computer Science and Engineering form West Bengal University of Technology, West Bengal, India. His field of interest includes Image Processing, Number Theory and Artificial Intelligence. He has published six papers in international Conference. His two research papers have been published in International journal.

Dr. Sanjoy Das is presently working as a Scientific Officer of Department of Engineering And Technological Studies at University of Kalyani, Kalyani, Nadia, West Bengal India-741235